Journal of Science Engineering Technology and Management Science
Volume 02, Issue 08, August 2025 *ISSN: 3049-0952*
www.jsetms.com *DOI: 10.64771/jsetms.2025.v02.i08.669-680*

AI DRIVEN DETECTION OF INJECTION ATTACKS IN API'S USING BIDIRECTIONAL RECURRENT NEURAL NETWORKS

Dr. A. Yashwanth Reddy[1], Sri Harsha Dixit[2], SV. Swathi[2], N. Shashank[2], K. Prashanth[2]
[1]Professor, [2]UG Student, [1,2] Department of Computer Science and Engineering (Data Science),
[1,2]Sree Dattha Group of Institutions, Sheriguda, Ibrahimpatnam, 501510, Telangana.

To Cite this Article

Dr. A. Yashwanth Reddy, Sri Harsha Dixit, SV. Swathi, N. Shashank, K. Prashanth, "Ai Driven Detection Of Injection Attacks In Api's Using Bidirectional Recurrent Neural Networks", Journal of Science Engineering Technology and Management Science, Vol. 02, Issue 08, August 2025,pp: 669-680, DOI: http://doi.org/10.64771/jsetms.2025.v02.i08.pp669-680

Submitted: 15-07-2025 *Accepted: 21-08-2025* *Published: 28-08-2025*

ABSTRACT

Injection attacks in APIs are a significant security concern, where malicious actors exploit vulnerabilities by inserting harmful code into API requests. These attacks can lead to unauthorized access, data breaches, and system compromises, ultimately affecting the integrity and confidentiality of applications. Historically, the detection of injection attacks in APIs has evolved alongside the increasing reliance on APIs in modern software development. Early detection methods primarily involved manual code reviews and pattern matching, which were often insufficient due to the complexity and variety of injection techniques. As APIs became more prevalent, the demand for automated and sophisticated detection mechanisms grew. Traditional systems for mitigating injection attacks have relied on input validation, parameterized queries, and the use of Web Application Firewalls (WAFs). These methods aim to prevent malicious inputs from being processed by the system. However, they frequently fall short in detecting complex or novel attack patterns, leaving systems vulnerable. The motivation for developing advanced detection systems arises from the limitations of traditional approaches. The increasing sophistication of attackers and the critical role of APIs in modern applications necessitate more robust and intelligent security measures. This research is driven by the need to enhance detection capabilities, reduce false positives, and adapt to emerging threats in real time. A core problem with traditional systems is their reactive nature and limited adaptability. They often require manual updates to recognize new attack vectors and struggle to detect obfuscated or zero-day attacks. Additionally, their reliance on predefined rules results in high false positive rates, where legitimate traffic is mistakenly flagged as malicious. To address these challenges, the proposed system leverages Bidirectional Recurrent Neural Networks (BRNNs) to detect injection attacks in APIs. BRNNs process input sequences in both forward and backward directions, capturing contextual information more effectively than unidirectional models. This approach enables the system to identify complex patterns associated with injection attacks, thereby improving detection accuracy and adaptability to evolving threats.

Keywords: Injection Attacks, API Security, Bidirectional Recurrent Neural Networks (BRNNs), Detection Accuracy, Automated Threat Detection

1.INTRODUCTION

This research focuses on detecting injection attacks in APIs using Bidirectional Recurrent Neural Networks (BiRNNs). Injection attacks—including SQL, XML, and JSON injections—pose significant threats to web applications and APIs, often resulting in data breaches and unauthorized access. By leveraging the capabilities of BiRNNs, the proposed system can effectively analyze sequential patterns in API requests, offering a powerful method for identifying and mitigating these attacks. The approach involves transforming API requests into TF-IDF vectors and utilizing deep learning to classify them as either normal or malicious. This ensures a higher level of security for modern web services and addresses the growing need for intelligent detection mechanisms in the digital landscape. The central challenge addressed by this research is the increasing sophistication of injection attacks targeting APIs, which frequently bypass traditional rule-based detection methods. Existing solutions often struggle to process large volumes of data or to adapt to the rapidly evolving techniques employed by attackers. Moreover, many conventional systems lack real-time processing capabilities, resulting in delays in detection and response. These shortcomings highlight the need for an advanced, scalable solution capable of detecting a wide range of injection attacks with high accuracy, speed, and adaptability.

The motivation behind this study stems from the rapid rise in API usage for web applications and microservices, which has in turn made APIs a major target for cyberattacks. Traditional detection methods often rely on static rules or signatures that attackers can easily circumvent. To address these gaps, this research harnesses the power of deep learning—particularly BiRNNs—to develop a dynamic, adaptive model capable of accurately detecting injection attacks. Enhancing the security of APIs is not only crucial for protecting sensitive information but also for maintaining user trust and ensuring the stability of digital services across various industries. Traditional systems for injection attack detection, such as signature-based tools, intrusion detection systems (IDS), and heuristic-based methods, have shown significant limitations. These tools typically depend on predefined patterns and manual rule sets, making them ineffective against zero-day vulnerabilities and novel injection techniques. Additionally, their inability to process large-scale data or understand the contextual relationships within API requests often leads to high false positive and false negative rates. These weaknesses underscore the need for a more intelligent and context-aware approach to threat detection.

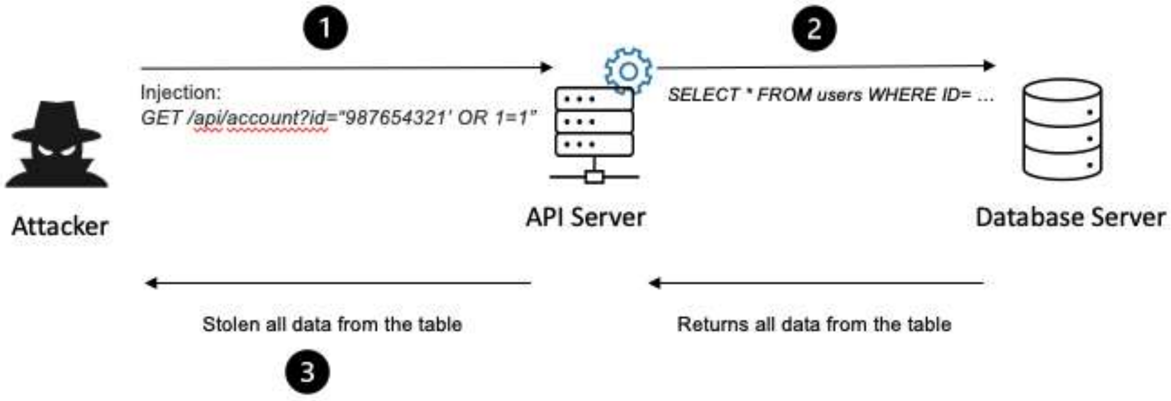

1
Injection:
GET /api/account?id="987654321' OR 1=1"
2
SELECT * FROM users WHERE ID= ...
Attacker
API Server
Database Server
Stolen all data from the table
Returns all data from the table
3

Fig.1: SQL Injection Attack via API Endpoint.

The objective of this research is to design and implement a robust, intelligent detection system for identifying injection attacks in API traffic using BiRNNs. By focusing on the sequential nature of API request data, the system aims to achieve high levels of precision, recall, and accuracy in distinguishing between benign and malicious activity. Another key aim is to ensure that the system can be deployed in real-world environments, maintaining effectiveness without imposing significant computational costs or latency, thus supporting both scalability and real-time application.

Given the growing reliance on APIs in various domains, securing them against injection attacks is more critical than ever. The dynamic nature of modern attack vectors renders traditional methods increasingly obsolete. There is a pressing need for advanced, adaptive solutions that can evolve with the threat landscape. This research directly addresses that need, offering a method that enhances the protection of sensitive data, ensures the integrity of systems, and fortifies the overall security posture of web-based services and applications.

The applications of this research are wide-ranging and impactful. It can protect APIs in e-commerce platforms from SQL injections, safeguard sensitive information in healthcare systems, and prevent unauthorized access in financial services. Social media platforms, enterprise microservices, and IoT ecosystems can all benefit from enhanced injection attack detection. Additionally, the system can be deployed for real-time threat monitoring in cloud-based environments and used to secure educational platforms handling student and administrative data. The versatility and effectiveness of the proposed approach make it a valuable tool for strengthening API security across diverse sectors.

2. LITERATURE SURVEY

Most cyber-physical system (CPS) applications are safety-critical; misbehavior caused by random failures or cyber-attacks can considerably restrict their growth. Thus, it is important to protect CPS from being damaged in this way [1]. Current security solutions have been well-integrated into many networked systems including the use of middle boxes, such as antivirus protection, firewall, and intrusion detection systems (IDS). A firewall controls network traffic based on the source or destination address. It alters network traffic according to the firewall rules. Firewalls are also limited to their knowledge of the hosts receiving the content and the amount of state available. An IDS is a type of security tool that scans the system for suspicious activity, monitors the network traffic, and alerts the system or network administrator [2]. In this context, a number of frameworks and mechanisms have been suggested in recent papers. In the project, we have considered SQL injection attacks that target the HTTP/HTTPS protocol, which aim to pass through the web application firewall (WAF) and obtain an unauthorized access to proprietary data. SQL injection belongs to the injection family of web attacks, wherein an attacker inserts inputs into a system to execute malicious statements. The victim system is usually not ready to process this input, typically resulting in data leakage and/or granting of unauthorized access to the attacker; in this case, the attacker can access and/or modify the data, affecting all aspects of security, including confidentiality, integrity, and data availability [3].

In an SQL injection, the attacker inserts an SQL statement into an exchange between a client and database server [3]. SQL (structured query language) is used to represent queries to database management systems (DBMSs). The maliciously injected SQL statement is designed to extract or modify data from the database server. A successful injection can result in authentication and bypass and changes to the database by inserting, modifying, and/or deleting data, causing data loss and/or destruction of the entire database. Furthermore, such an attack could overrun and execute commands on the hosted operating system, typically leading to more serious consequences [4]. Thus, SQL injection attacks present aserious threats to organizations. A variety of research has been undertaken to address this threat, presenting various artificial intelligence (AI)techniques for detection of SQL injection attacks using machine learning and deep learning models [5]. AI techniques to facilitate the detection of threats are usually implemented via learning from historical data representing an attack and/or normal data. Historical data are useful for learning, in order to recognize patterns of attacks, understanding detected traffic, and even predicting future attacks before they occur [6].

In some research, injecting a code using 'OR' followed by a TRUE statement, such as 1 = 1 is called "tautology" [**7**]. Methods other than tautology can be used, such as when an attacker intentionally injects an incorrect query to force the database server to return a default error page, which might contain valuable information that could help an attacker to understand the database to form a more advance attack [**7**]. The SQL syntax "UNION" can also be used to extract information, in addition to many other methods based on the same idea, of misusing SQL syntax to extract or even update the data in the targeted database. This is how SQL injection works; the question then becomes: how does one detect this type of attack using deep learning methods? Deep learning is a machine learning and artificial intelligence method. It can be used to support the detection of SQL injection attacks by training a classifier to achieve the ability to recognize and therefore detect an attack. The classifier is trained using deep learning models and can be used to classify new data, such as traffic or data in log files. If the classifier is passive, it will alert the administrator; if it is active, it will prevent data from passing to the database server. The classifier can be trained to recognize and detect SQL injection attacks using three different learning methods [8]. In the section, four published systematic reviews were considered. Newer systematic reviews typically include both recent and older studies in the area under investigation. Therefore, all of the papers we considered were relatively recent. The first was published in 2017 [9] and it covered previous primary studies on SQL injection attacks, techniques, and tools.

In [10], forty-six primary studies were analyzed related to SQL injection attacks, tools, and techniques, in addition to the impact of the attack. We adapted the same methodology as that used in [11] due to its comprehensiveness and because it achieves satisfying results, in addition, this research was similar to that in [12] in terms of objectives, ideas, and the area of research. Qiu et al. [13] provided a comprehensive review of using artificial intelligence in attacking and defending against security attacks, concentrating on the training and testing stages. In their study, they sorted technologies and applications of adversarial attacks in terms of natural language processing, cyberspace security, computer vision, and the physical world. Furthermore, the authors considered defense strategies in their research and proposed methods to deal with specific types of adversarial attack.

Martins et al. [14] explored more than 15 papers that applied adversarial machine learning techniques used in intrusion and malware detection models. In their study, the authors summarized the most common adversarial attacks and defense mechanisms for intrusion and malware detection.

Muslihi et al. [15] conducted a review of more than 14 studies published using deep learning methods to detect SQL injection attacks, including CNN, LSTM, DBN, MLP, and Bi-LSTM. They also provided a comparison of methods in terms of objectives, techniques, features, and datasets. Muhammad et al. [16] reviewed and analytically evaluated the methods and tools that are commonly used to detect and prevent SQL injection attacks, considering a total of 82 studies. Their review results showed that most researchers focused on proposing approaches to detect and mitigate SQL injection attacks (SQLIAs) rather than evaluating the effectiveness of existing SQLIA detection methods.

3.PROPOSED SYSTEM

The proposed system for detecting injection attacks in APIs employs advanced machine learning techniques, specifically Gated Recurrent Units (GRUs), to enhance detection accuracy and computational efficiency. The overall approach follows a structured pipeline, beginning with the collection of a comprehensive dataset consisting of both benign and malicious API requests. This dataset, stored in CSV format, is the foundation of the system, as the quality and diversity of the data significantly impact the performance of the machine learning model. Effective preprocessing is applied to the raw data to ensure consistency and suitability for analysis. This includes removing null values, performing label encoding to convert categorical classes into numerical representations, and applying TF-IDF vectorization to extract meaningful textual features from API requests while removing stop words to reduce noise.

Once the data is preprocessed, it undergoes further transformation to prepare it for input into deep learning models. The data is shuffled to eliminate any ordering bias and then split into training and testing sets, typically in an 80:20 ratio. Because recurrent neural networks (RNNs) like GRUs require 3D input, the 2D TF-IDF vectors are reshaped to a compatible format where the number of time steps is set to one. This reshaping enables the GRU model to interpret each API request as a sequence, allowing it to detect both local and global patterns within the data.

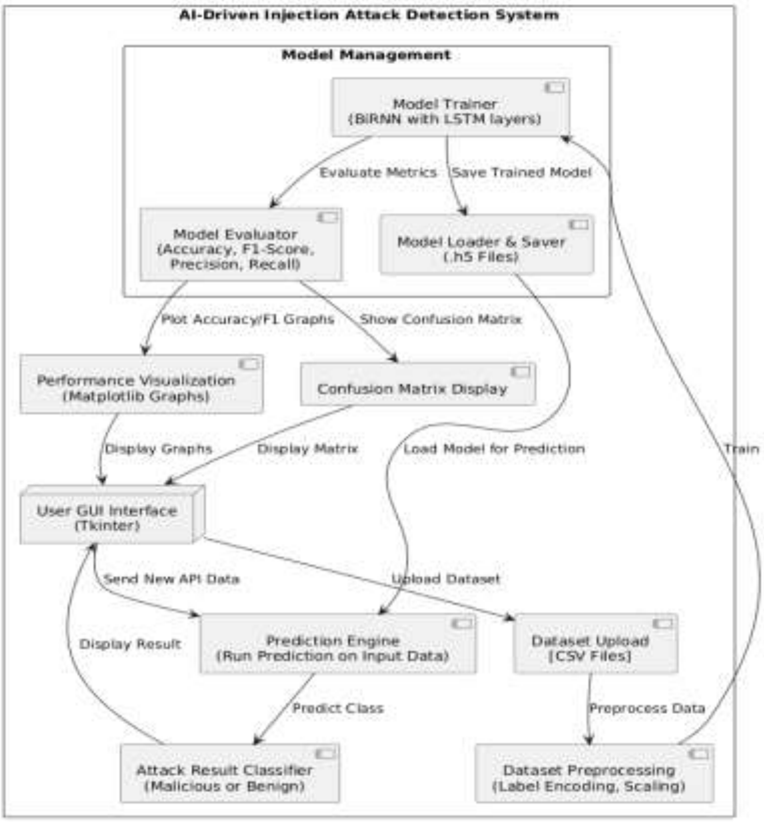

AI-Driven Injection Attack Detection System
Model Management
Model Trainer (BiRNN with LSTM layers)
Evaluate Metrics
Save Trained Model
Model Evaluator (Accuracy, F1-Score, Precision, Recall)
Model Loader & Saver (.h5 Files)
Plot Accuracy/F1 Graphs
Show Confusion Matrix
Performance Visualization (Matplotlib Graphs)
Confusion Matrix Display
Display Graphs
Display Matrix
Load Model for Prediction
Train Model
User GUI Interface (Tkinter)
Send New API Data
Upload Dataset
Display Result
Prediction Engine (Run Prediction on Input Data)
Dataset Upload (CSV Files)
Predict Class
Preprocess Data
Attack Result Classifier (Malicious or Benign)
Dataset Preprocessing (Label Encoding, Scaling)

Fig.2: Proposed block diagram of detection of injection attacks in API's.

The model-building process begins with a Vanilla RNN used as a baseline, followed by the implementation of a GRU-based architecture for enhanced performance. GRUs are chosen due to their ability to address the limitations of traditional RNNs, such as the vanishing gradient problem, which hinders the learning of long-term dependencies. The GRU model comprises several key components, including a GRU layer that captures sequential dependencies in the API data using update and reset gates, a dense layer with ReLU activation to process the output from the GRU, and a dropout layer that helps prevent overfitting during training. The final output layer uses softmax activation to produce probability distributions across multiple classes, such as benign, SQL injection, cross-site scripting (XSS), and remote code execution (RCE).

The model is trained using categorical cross-entropy as the loss function, optimized via the Adam optimizer for adaptive learning. The training process leverages Backpropagation Through Time (BPTT) to iteratively minimize error and fine-tune model weights. During testing, the trained GRU model is evaluated on unseen API request sequences to assess its generalization capabilities. The test data is preprocessed similarly to the training data to ensure consistency. For each input, the model predicts the probability of each class, selecting the one with the highest score as the final prediction.

Evaluation metrics play a crucial role in validating the system's effectiveness. The predicted labels are compared with actual labels from the test set to calculate performance indicators such as accuracy, precision, recall, and F1-score. These metrics provide insights into the model's ability to correctly identify true attacks, minimize false positives, and maintain a balance between sensitivity and specificity. A confusion matrix is also generated to visualize the distribution of correct and incorrect predictions across different attack types, helping identify areas where the model excels or needs improvement.

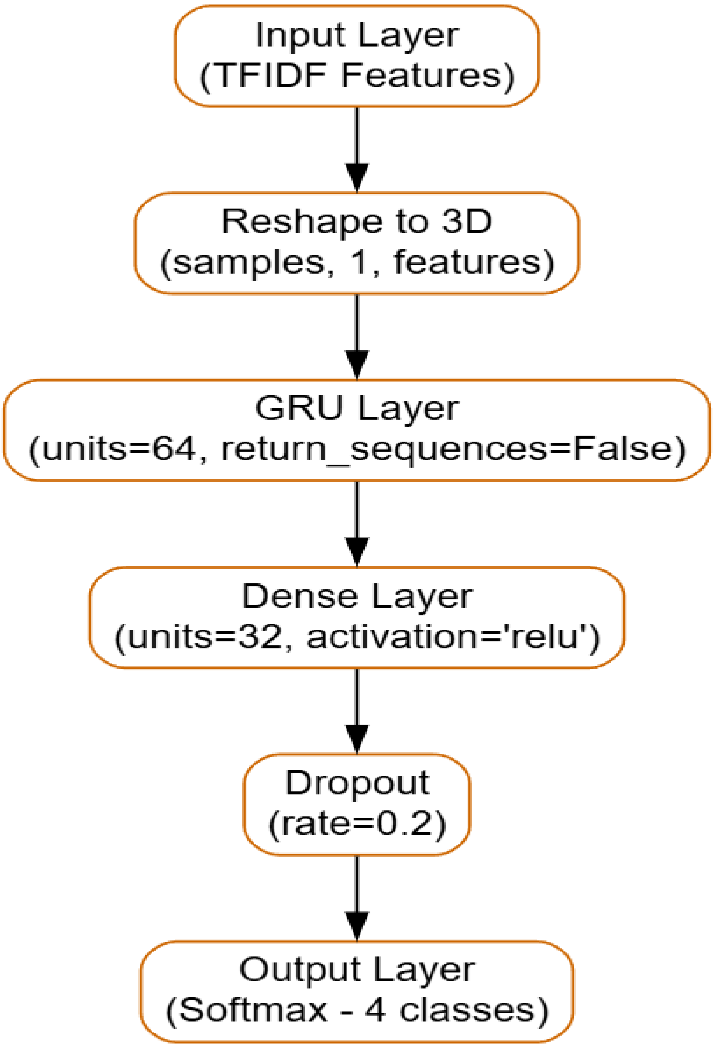


Fig.3: Workflow of proposed GRU model.

The GRU-based system offers several advantages for API injection detection. Its ability to learn sequential API behavior allows it to detect complex attacks that unfold across multiple calls. Unlike LSTMs, GRUs are more compact and require fewer computational resources while still capturing critical temporal dependencies. The model is resistant to the vanishing gradient problem, enabling it to retain relevant context over longer sequences. It also eliminates the need for manual feature engineering by learning temporal patterns directly from the raw input. Furthermore, the system supports real-time detection, making it suitable for deployment in environments that monitor live API traffic or logs. Compared to simple RNNs, GRUs demonstrate better generalization and are more effective in distinguishing nuanced attack patterns from legitimate API behavior, ultimately enhancing the security posture of modern web applications.

4. RESULTS AND DISCUSSION

The implementation of the proposed injection attack detection system begins with the integration of essential Python libraries. These include Tkinter for creating the graphical user interface (GUI), Pandas and NumPy for efficient data handling and numerical operations, and Matplotlib along with Seaborn for generating informative visualizations. For machine learning and deep learning tasks, libraries such as Keras and Scikit-learn are used, while NLTK handles natural language processing components like stop word removal. Additionally, Pickle is employed to save and load model training history and weights, enabling efficient experimentation and reproducibility.

The user interface is developed using Tkinter, with a main window that includes various interactive components such as buttons, labels, and text areas. These enable users to upload datasets, perform preprocessing, train models, visualize results, and make predictions through an intuitive GUI. Upon uploading a dataset in CSV format containing API request strings and their associated labels (e.g., Normal, SQL Injection, XML/JSON Injection), the system displays the data in a scrollable text box and plots a class distribution bar graph to provide immediate insights into the dataset's balance.

Data preprocessing involves converting the text labels into integer formats, such as 0 for normal requests and other values for specific injection types. Textual API requests are transformed into numerical vectors using the Term Frequency-Inverse Document Frequency (TF-IDF) technique. During this step, English stop words are removed to reduce noise and improve the quality of feature extraction. The resulting TF-IDF vectors are reshaped into a format suitable for input into recurrent neural network models and then shuffled and split into training and testing subsets, typically using an 80:20 ratio.

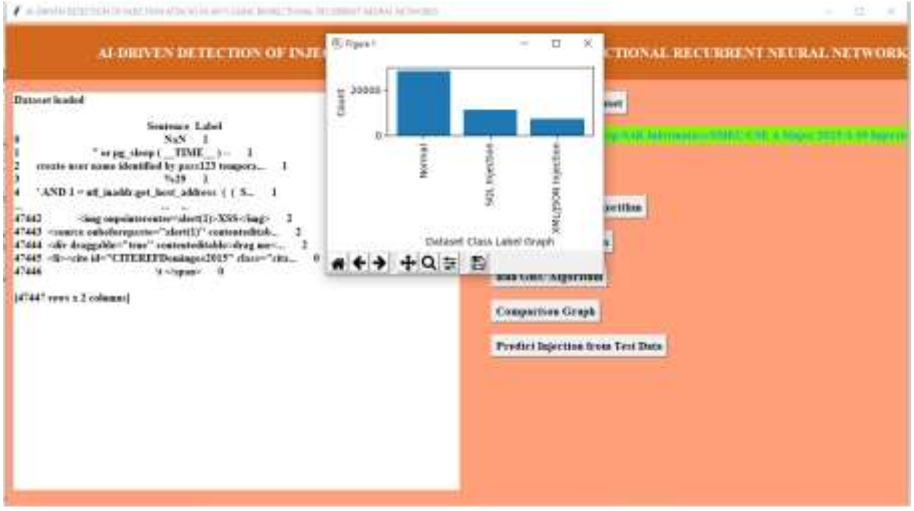
Model training begins with a Vanilla RNN implementation as the baseline model. The architecture includes an input layer aligned with the shape of the processed dataset, a dropout layer to mitigate overfitting, a dense hidden layer using the ReLU activation function, and a softmax output layer for multi-class classification. The model is compiled with a categorical cross-entropy loss function and the Adam optimizer and is either trained on the dataset or initialized with pre-saved weights for faster experimentation.

Subsequent models, including LSTM and GRU, are implemented following a similar architecture, replacing the basic RNN layer with more advanced variants. LSTM is employed to better capture long-term dependencies in sequential data, while GRU offers a simplified yet efficient alternative to LSTM with fewer parameters and comparable performance. Both models use checkpointing to save weights during training and allow for easy reloading in future sessions. Once training is complete, all models are evaluated on the test dataset. Predictions are generated and compared against true labels to compute key performance metrics such as accuracy, precision, recall, and F1-score. Confusion matrices are also generated and visualized using Seaborn heatmaps to illustrate model performance in distinguishing between different attack types. These metrics are then aggregated and plotted in a comparative bar chart to clearly show the relative performance of Vanilla RNN, LSTM, and GRU, helping users identify the most effective architecture for detecting injection attacks.

The system also supports real-time predictions on new data. Users can upload unseen datasets, which are preprocessed using the same TF-IDF vectorizer and reshaped for model compatibility. The pre-trained GRU model is then used to classify each entry, and predictions are displayed in the GUI. Throughout the entire process, the GUI provides live feedback on the system's activities, including dataset details, training progress, evaluation results, and prediction outputs through a scrollable text widget. Graphs and confusion matrices are shown in pop-up windows, offering a comprehensive and interactive user experience. The dataset used in this project is carefully structured to facilitate the detection of API-based injection attacks, particularly SQL injection and Cross-Site Scripting (XSS). It contains two primary columns: 'Sentence' and 'Label.' The 'Sentence' column consists of various textual inputs representing API calls or user queries, which may contain SQL code, HTML tags, or scripts designed to exploit system vulnerabilities. These inputs range from benign requests to malicious payloads crafted to bypass security measures and execute unauthorized actions. Examples of malicious entries include SQL payloads like " or pg_sleep(__TIME__) -- intended to manipulate database queries, and XSS vectors like <img onpointerenter=alert(1)>XSS</img> designed to execute JavaScript in user browsers.

The 'Label' column indicates whether each sentence is safe or malicious, typically using numerical values: 0 for benign inputs, 1 for SQL injections, and 2 for XSS attacks. This labeling scheme supports the classification task by serving as the ground truth for training and evaluating the model. The dataset is intentionally diverse, capturing a wide spectrum of attack vectors to ensure the robustness and generalization capability of the detection system. By combining clear sentence representations with accurate labeling, the dataset provides a strong foundation for training deep learning models to identify and mitigate injection attacks in real-world API environments.

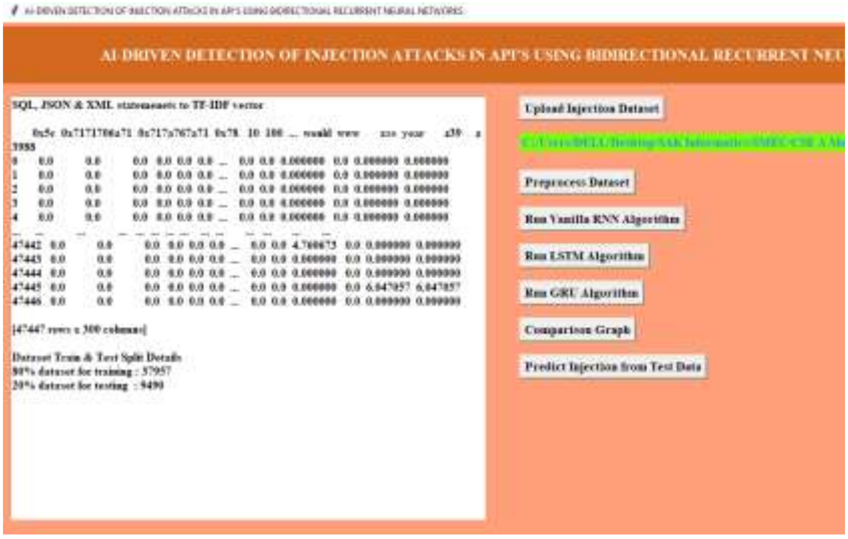
Results Description


AI-DRIVEN DETECTION OF INJE
CTIONAL RECURRENT NEURAL NETWORK
Figure 1
Count
20000
0
Normal
SQL Injection
XML/JSON Injection
Dataset Class Label Graph
Comparison Graph
Predict Injection from Test Data

```
Dataset loaded

                                        Sentence  Label
0                                            NaN      1
1                   " or pg_sleep ( __TIME__ )--      1
2   create user name identified by pass123 tempora...  1
3                                            %29      1
4       ' AND 1 = utl_inaddr.get_host_address ( ( S...  1
...                                          ...    ...
47442         <img onpointerenter=alert(1)>XSS</img>      2
47443  <source onbeforecopy="alert(1)" contenteditab...   2
47444  <div draggable="true" contenteditable>drag me<...  2
47445  <li><cite id="CITEREFDomingos2015" class="cita...  0
47446                                   \t </span>      0

[47447 rows x 2 columns]
```

Fig.4: Illustration of GUI interface showing data preprocessing.

The Figure 4 shows the user interface of a desktop application designed for detecting injection attacks in APIs. The user uploads a dataset of API requests or code samples using the "Upload Injection Dataset" button. Presents a bar chart visualizing the distribution of different attack types and normal instances in the loaded dataset. This provides an overview of the dataset's composition.

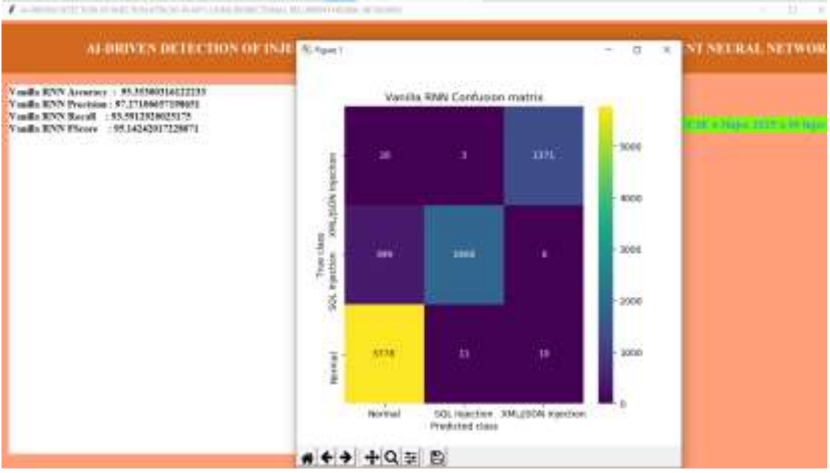

AI-DRIVEN DETECTION OF INJECTION ATTACKS IN API'S USING BIDIRECTIONAL RECURRENT NEURAL NETWORKS
AI DRIVEN DETECTION OF INJECTION ATTACKS IN API'S USING BIDIRECTIONAL RECURRENT NEU
Upload Injection Dataset
Preprocess Dataset
Run Vanilla RNN Algorithm
Run LSTM Algorithm
Run GRU Algorithm
Comparison Graph
Predict Injection from Test Data

```
SQL, JSON & XML statements to TF-IDF vector

       0x5c  0x7171706a71  0x717a767a71  0x78  10  100  ...  would  www      xss  year      z39    z
3988
0       0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  0.000000  0.0  0.000000  0.000000
1       0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  0.000000  0.0  0.000000  0.000000
2       0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  0.000000  0.0  0.000000  0.000000
3       0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  0.000000  0.0  0.000000  0.000000
4       0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  0.000000  0.0  0.000000  0.000000
...     ...           ...           ...   ...  ...  ...  ...  ...  ...       ...  ...       ...       ...
47442   0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  4.760673  0.0  0.000000  0.000000
47443   0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  0.000000  0.0  0.000000  0.000000
47444   0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  0.000000  0.0  0.000000  0.000000
47445   0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  0.000000  0.0  6.647057  6.647057
47446   0.0           0.0           0.0   0.0  0.0  0.0  ...  0.0  0.0  0.000000  0.0  0.000000  0.000000

[47447 rows x 300 columns]

Dataset Train & Test Split Details
80% dataset for training : 37957
20% dataset for testing  : 9490
```

Fig.5: Illustration of GUI interface showing data splitting.

The Figure 5 describes the data splitting process, a crucial step in machine learning model development. Let's break down what it means: This clearly indicates that the data has been divided into two sets: a training set and a testing set. This is standard practice to evaluate how well a machine learning model generalizes to unseen data.

In this implementation, the dataset is divided into two subsets to facilitate effective model training and evaluation. A total of 80% of the dataset, comprising 37,957 data samples, is allocated for training. This training set is used to teach the machine learning model to recognize patterns and relationships within the data, allowing it to learn the distinctions between benign and malicious API requests. The remaining 20% of the dataset, consisting of 9,490 samples, is reserved for testing. This testing set plays a crucial role in evaluating the model's performance on unseen data, helping to determine its accuracy, robustness, and generalization capability. By assessing how well the model performs on the test set, developers can gain insight into its effectiveness in real-world scenarios and ensure it does not overfit to the training data



Vanilla RNN Accuracy : 95.35300316122233
Vanilla RNN Precision : 97.17186057159051
Vanilla RNN Recall : 93.59123280025175
Vanilla RNN FScore : 95.14242017228871

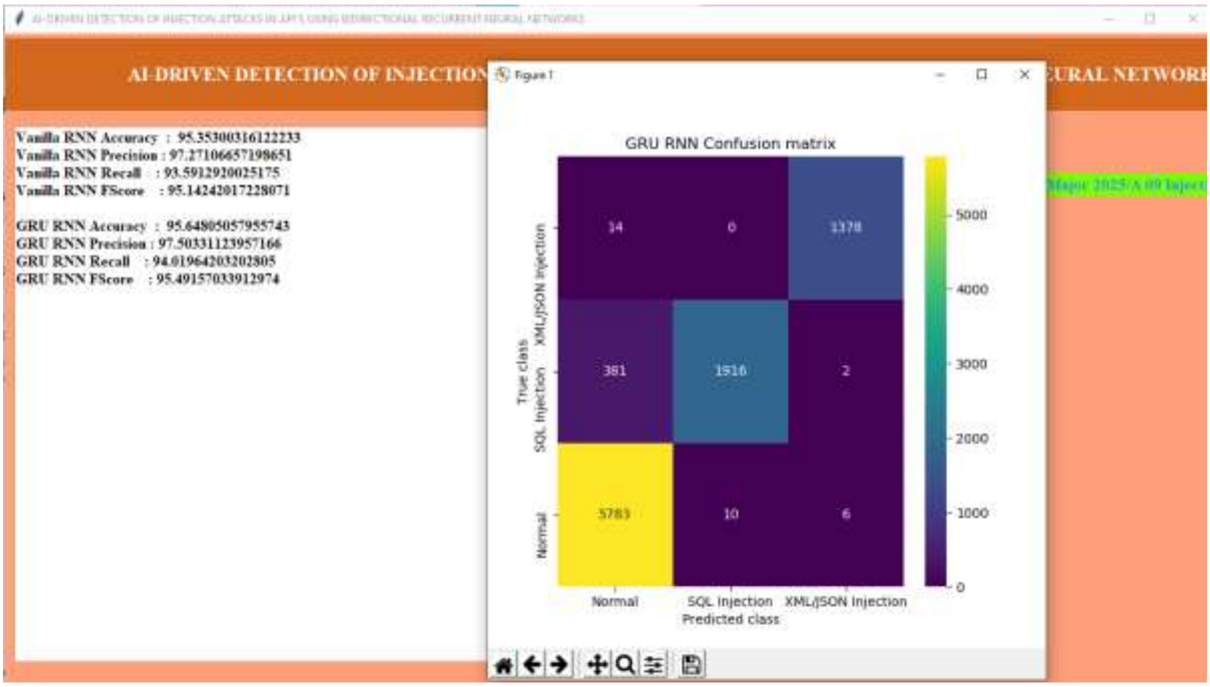
.

Fig.6: Confusion matrix obtained using RNN model.

displays the performance evaluation of a machine learning model, specifically a "Vanilla RNN" (Recurrent Neural Network), likely used for a classification task. Let's break down the information presented:

The performance of the Vanilla RNN model is evaluated using standard classification metrics and a confusion matrix. The model achieved an impressive overall accuracy of 95.35%, indicating that it correctly classified approximately 95.35% of all input instances in the dataset. The precision of the model stands at 97.17%, reflecting its ability to make highly accurate predictions, particularly for the "Normal" class—meaning that when the model predicts an instance as "Normal," it is correct in the majority of cases. The recall value is 93.59%, which shows the model's effectiveness in identifying actual instances of each class, particularly its capability to recognize most of the "Normal" inputs. The F1-score, a balanced metric that combines both precision and recall, is 95.14%, signifying strong overall performance, especially in scenarios with imbalanced class distributions.

The confusion matrix further breaks down the model's predictive performance by showing how many instances of each true class were correctly or incorrectly classified. The matrix is structured with actual classes represented along the rows and predicted classes along the columns. The diagonal cells—representing correct classifications—highlight that the model correctly identified 5,778 instances of "Normal," 1,900 instances of "SQL Injection," and 1,371 instances of "XML/JSON Injection." These diagonal values demonstrate the model's strong classification ability. However, the off-diagonal cells reveal some misclassifications. For instance, 399 instances of "SQL Injection" were incorrectly labeled as "Normal," and 11 "Normal" instances were misclassified as "SQL Injection." These errors, though relatively minimal, indicate areas where the model could be further fine-tuned to enhance its discrimination between closely related or overlapping input patterns. Overall, the performance metrics and confusion matrix collectively validate the Vanilla RNN model's robustness in detecting injection attacks while also providing insight into specific areas for improvement.



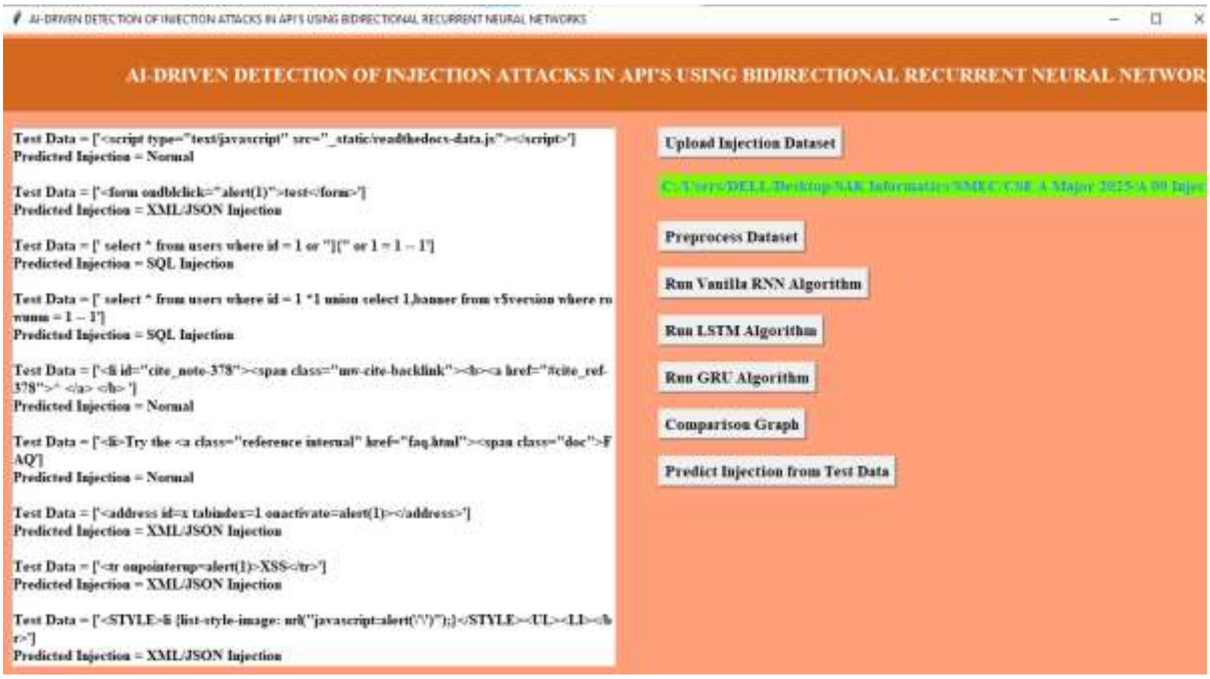
```
Vanilla RNN Accuracy  :  95.35300316122233
Vanilla RNN Precision : 97.27106657198651
Vanilla RNN Recall    : 93.5912920025175
Vanilla RNN FScore    : 95.14242017228071

GRU RNN Accuracy  :  95.64805057955743
GRU RNN Precision : 97.50331123957166
GRU RNN Recall    : 94.01964203202805
GRU RNN FScore    : 95.49157033912974
```

Fig.7: Confusion matrix obtained using GRU model

displays the performance evaluation of a machine learning model, specifically a "GRU" (Gated Recurrent Unit), likely used for a classification task. Let's break down the information presented:

The performance of the GRU-based RNN model demonstrates a slight but meaningful improvement over the Vanilla RNN, showcasing its effectiveness in detecting injection attacks in API requests. The GRU model achieved an accuracy of 95.65%, indicating that it correctly classified approximately 95.65% of all instances in the dataset. This marks a small but notable increase in performance compared to the Vanilla RNN. The precision of the GRU model is 97.50%, reflecting a high level of confidence in its predictions—when the model assigns a class label, it is highly likely to be correct. The recall value of 94.02% indicates the model's strong ability to correctly identify a majority of actual class instances, which is crucial in a security context where missing a malicious input can have serious consequences. The F1-score, which balances both precision and recall, stands at 95.49%, slightly higher than the Vanilla RNN's score, suggesting that the GRU model offers better overall classification reliability.

The confusion matrix for the GRU model provides a detailed view of its predictive accuracy across specific classes. Along the diagonal, where correct classifications are recorded, the model correctly identified 5,783 instances of "Normal" API requests, 1,916 instances of "SQL Injection," and 1,378 instances of "XML/JSON Injection." These results indicate the model's strong capability to distinguish between benign and malicious traffic. However, the matrix also highlights several misclassifications. Notably, 381 instances of "SQL Injection" were incorrectly labeled as "Normal," which could pose a risk in a real-world scenario. Additionally, 10 "Normal" requests were misclassified as "SQL Injection," and 14 "XML/JSON Injection" instances were mislabeled as "Normal." Other minor errors include 2 "SQL Injection" samples predicted as "XML/JSON Injection" and 6 "Normal" instances mistakenly classified as "XML/JSON Injection." Despite these misclassifications, the GRU model demonstrates improved precision and recall over the Vanilla RNN, making it a more reliable and robust choice for detecting injection attacks in API environments.


AI-DRIVEN DETECTION OF INJECTION ATTACKS IN API'S USING BIDIRECTIONAL RECURRENT NEURAL NETWORKS
AI-DRIVEN DETECTION OF INJECTION ATTACKS IN API'S USING BIDIRECTIONAL RECURRENT NEURAL NETWOR
Upload Injection Dataset
C:/Users/DELL/Desktop/SAK Informatics/SMEC/CSE A Major 2025/A 08 Injec
Preprocess Dataset
Run Vanilla RNN Algorithm
Run LSTM Algorithm
Run GRU Algorithm
Comparison Graph
Predict Injection from Test Data

```
Test Data = ['<script type="text/javascript" src="_static/readthedocs-data.js"></script>']
Predicted Injection = Normal

Test Data = ['<form ondblclick="alert(1)">test</form>']
Predicted Injection = XML/JSON Injection

Test Data = [' select * from users where id = 1 or "]{" or 1 = 1 -- 1']
Predicted Injection = SQL Injection

Test Data = [' select * from users where id = 1 *1 union select 1,banner from v$version where ro
wnum = 1 -- 1']
Predicted Injection = SQL Injection

Test Data = ['<li id="cite_note-378"><span class="mw-cite-backlink"><b><a href="#cite_ref-
378">^ </a> </b> ']
Predicted Injection = Normal

Test Data = ['<li>Try the <a class="reference internal" href="faq.html"><span class="doc">F
AQ']
Predicted Injection = Normal

Test Data = ['<address id=x tabindex=1 onactivate=alert(1)></address>']
Predicted Injection = XML/JSON Injection

Test Data = ['<tr onpointerup=alert(1)>XSS</tr>']
Predicted Injection = XML/JSON Injection

Test Data = ['<STYLE>li {list-style-image: url("javascript:alert('\')");}</STYLE><UL><LI></b
r>']
Predicted Injection = XML/JSON Injection
```

Fig.8: Illustration of GUI interface showing predicted outcomes.

The Figure 8 displays the results of an injection attack detection system on various test data inputs. The system analyzes each input, attempting to identify if it contains a potential injection attack and, if so, the type of attack. The "Predicted Injection" column shows the model's classification for each "Test Data" entry. A variety of inputs are tested, including script injections, form manipulations, SQL injection attempts, and others containing HTML elements or JavaScript code. The model classifies the inputs as either "Normal" (no attack detected), "SQL Injection," or "XML/JSON Injection," correctly identifying several SQL and XML/JSON injection attempts while also classifying benign inputs as "Normal." However, some potentially malicious inputs, like those with script tags or JavaScript alerts, are also classified as "Normal" or "XML/JSON Injection" when they might represent other types of attacks (e.g., Cross-Site Scripting - XSS), indicating potential areas for improvement in the model's detection capabilities.

5. CONCLUSION

The implementation of a Gated Recurrent Unit (GRU)-based model for detecting injection attacks in APIs marks a significant advancement in cybersecurity. By leveraging the GRU's ability to capture sequential dependencies, the model effectively identifies complex patterns associated with SQL injection and Cross-Site Scripting (XSS) attacks. A comprehensive dataset comprising both malicious and benign API requests enabled robust training and evaluation, resulting in a model capable of accurately distinguishing between normal and anomalous behaviors. This approach addresses the limitations of traditional detection methods by providing a more adaptive and intelligent solution to evolving cyber threats. The success of this model highlights the potential of integrating advanced deep learning techniques into cybersecurity frameworks, thereby enhancing the resilience of API-driven applications against sophisticated injection attacks.

REFERENCES

[1]. Han, S.; Xie, M.; Chen, H.-H.; Ling, Y. Intrusion Detection in Cyber-Physical Systems: Techniques and Challenges. *IEEE Syst. J.* 2014, *8*, 1049–1059. [Google Scholar] [CrossRef]

[2]. Mishra, P.; Varadharajan, V.; Tupakula, U.; Pilli, E.S. A Detailed Investigation and Analysis of using Machine Learning Techniques for Intrusion Detection. *IEEE Commun. Surv. Tutor.* 2018, *21*, 686–728. [Google Scholar] [CrossRef]

[3]. Charles, M.J.; Pfleeger, P.; Pfleeger, S.L. *Security in Computing*, 5th ed.; Springer: Berlin/Heidelberg, Germany, 2004. [Google Scholar]

[4]. Son, S.; McKinley, K.S.; Shmatikov, V. Diglossia: Detecting code injection attacks with precision and efficiency. *Proc. ACM Conf. Comput. Commun. Secur.* 2013, *2*, 1181–1191. [Google Scholar] [CrossRef]

[5]. Yan, R.; Xiao, X.; Hu, G.; Peng, S.; Jiang, Y. New deep learning method to detect code injection attacks on hybrid applications. *J. Syst. Softw.* 2018, *137*, 67–77. [Google Scholar] [CrossRef]

[6]. Vähäkainu, P.; Lehto, M. Artificial intelligence in the cyber security environment. In Proceedings of the 14th International Conference on Cyber Warfare and Security, ICCWS 2019, Stellenbosch, South Africa, 28 February–1 March 2019; pp. 431–440. [Google Scholar]

[7]. Satapathy, S.C.; Govardhan, A.; Raju, K.S.; Mandal, J.K. SQL Injection Detection and Correction Using Machine Learning Techniques. *Adv. Intell. Syst. Comput.* 2015, *337*, 435–442. [Google Scholar] [CrossRef]

[8]. Marashdeh, Z.; Suwais, K.; Alia, M. A Survey on SQL Injection Attacks: Detection and Challenges. In Proceedings of the 2021 International Conference on Information Technology (ICIT), Amman, Jordan, 14–15 July 2021; pp. 957–962. [Google Scholar] [CrossRef]

[9]. Faker, S.A.; Muslim, M.A.; Dachlan, H.S. A systematic literature review on sql injection attacks techniques and common exploited vulnerabilities. *Int. J. Comput. Eng. Inf. Technol.* 2017, *9*, 284–291. [Google Scholar]

[10]. Qiu, S.; Liu, Q.; Zhou, S.; Wu, C. Review of artificial intelligence adversarial attack and defense technologies. *Appl. Sci.* 2019, *9*, 909. [Google Scholar] [CrossRef]

[11]. Martins, N.; Cruz, J.M.; Cruz, T.; Abreu, P.H. Adversarial Machine Learning Applied to Intrusion and Malware Scenarios: A Systematic Review. *IEEE Access* 2020, *8*, 35403–35419. [Google Scholar] [CrossRef]

[12]. Muslihi, M.T.; Alghazzawi, D. Detecting SQL Injection on Web Application Using Deep Learning Techniques: A Systematic Literature Review. In Proceedings of the 2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE), Surabaya, Indonesia, 3–4 October 2020. [Google Scholar] [CrossRef]

[13]. Aliero, M.S.; Qureshi, K.N.; Pasha, M.F.; Ghani, I.; Yauri, R.A. Systematic Review Analysis with SQLIA Detection and Prevention Approaches. *Wirel. Pers. Commun.* 2020, *112*, 2297–2333. [Google Scholar] [CrossRef]

[14]. Hasan, M.; Tarique, M. Detection of SQL Injection Attacks: A Machine Learning Approach. In Proceedings of the 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates, 19–21 November 2019. [Google Scholar]

[15]. Gao, H.; Zhu, J.; Liu, L.; Xu, J.; Wu, Y.; Liu, A. Detecting SQL Injection Attacks Using Grammar Pattern Recognition and Access Behavior Mining. In Proceedings of the 2019 IEEE International Conference on Energy Internet (ICEI), Nanjing, China, 27–31 May 2019. [Google Scholar] [CrossRef]

[16]. Gandhi, N. A CNN-BiLSTM based Approach for Detection of SQL Injection Attacks. In Proceedings of the 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, United Arab Emirates, 17–18 March 2021; pp. 378–383